1990

Efficient Detection of Quasiperiodicities in Strings

Alberto Apostolico

Andrzej Ehrenfeucht

Report Number:
90-1048

EFFICIENT DETECTION OF QUASIPERIODICITIES IN STRINGS

Alberto Apostolico
Andrzej Ehrenfeucht

CSD-TR-1048
November 1990

Efficient Detection of Quasiperiodicities in Strings*

Alberto Apostolico
Department of Computer Science, Purdue University
West Lafayette, IN 47907
and
Dipartimento di Matematica Pura e Applicata, Università de L'Aquila
L'Aquila, Italy

and

Andrzej Ehrenfeucht
Department of Computer Science, University of Colorado at Boulder
Boulder, Co. 80301

(August 1990, Revised October 1990)
Fibonacci Report 90.5
(October 1990)

Abstract: A string z is *quasiperiodic* if there is a second string $w \neq z$ such that the occurrences of w in z cover z entirely, i.e., every position of z falls within some occurrence of w in z. It is shown here that all maximal quasiperiodic substrings of a string x of n symbols can be detected in time $O(n \log^2 n)$.

Key words: combinatorial algorithms on words, string matching, avoidable regularities, squares and repetitions in a string, quasiperiodicities and quasiperiods, superprimitive strings, suffix trees.

AMS subject classification: 68C25

* This research was supported, through the Leonardo Fibonacci Institute, by the Istituto Trentino di Cultura, Trento, Italy. Additional support for A. Apostolico was provided in part by the National Research Council of Italy, by NFS Grant CCR-89-00305, by NIH Library of Medicine Grant R01 LM05118, by AFOSR Grant 90-0107, and by NATO Grant CRG900293.

1. INTRODUCTION

Periodicities and other regularities in strings represent a pervasive notion in many areas of Science, e.g., Combinatorics, Theory of Probability and Stochastic Processes, Symbolic Dynamics, System Theory, Molecular Biology, etc. In Computer Science these notions are encountered in Coding and Automata Theory, Formal Languages Theory, Data Compression, etc. A typical regularity that might affect an assigned string x is a *square*, i.e., a subpattern of x consisting of two consecutive instances of the same string (e.g., cbcb is a square in abcbcbabb, and so is bb). Squares in strings were first studied by Axel Thue [19, 20] early in this century. Thue discovered that, with an alphabet of more than two characters, one can build indefinitely long 'square-free' strings, i.e., strings having no squares as substrings. Hence, squares are *avoidable* [14] regularities in strings.

Since the work of Thue, a substantial body of literature has been developed on the subject. In particular, the problem has been approached of testing the square-freedom of a string and/or detecting and counting all squares and repetitions in a string. This problem is relevant to a variety of applications, some of which are listed in [15]. In addition, squares and repetitions play a significant role in the computation of some special substring statistics for a string [7].

There are optimal, linear-time sequential algorithms for testing the square-freedom of a string over a bounded alphabet [10, 16]. A fast and elegant square-freedom test using fingerprinting techniques was given in [18]. More recently, the problem has been studied also in the framework of parallel computation on a RAM with n processors. The CRCW algorithm in [11] takes $O(\log^2 n)$ time and linear space or $O(\log n)$ time and quadratic space. The CRCW algorithm in [3] takes $O(\log n)$ time and linear space. The fastest sequential algorithms [6, 9, 15] detect all squares in $O(n \log n)$ time. As shown in [9], there can be $\Theta(n \log n)$ distinct positioned squares in a string x of n symbols. A notable example of such classes of strings is offered by the *Fibonacci words* which are defined recursively as follows: $f_0 = a; f_1 = b$ and, for $i > 1$, $f_i = f_{i-1} f_{i-2}$. (Fibonacci words not only have $\Theta(n \log n)$ distinct positioned squares, but also $\Theta(n \log n)$ distinct square substrings [4].) Thus, the algorithms in [6, 9, 15] are optimal, and the algorithm in [3] achieves optimal *speed-up*.

In this paper, we introduce and study another form of regularity in strings that we call quasiperiodicity. A string z is *quasiperiodic* if there is a second string $w \neq z$ such that every position of z falls within some occurrence of w in z. For example, the string $z = abaababababaaba$ is quasiperiodic, since it can be obtained by the concatenation and superposition of 5 instances of $w = aba$. It is clear that a string contains some quasiperiodic substring only if it contains a square. Since squares are avoidable regularities in strings, so are also the quasiperiodicities. Here, we show that all maximal quasiperiodic substrings

of a string x of n symbols can be detected in time $O(n \log^2 n)$ and linear auxiliary space. Informally, a quasiperiodic substrings z of x is maximal if no extension of z could be covered by either the same word w covering z or by an extension wa of w.

The paper is organized as follows. In the next section, we recall some basic facts of combinatorics on words. Section 3 contains theoretical developments that subtend the criteria used by the algorithm. Sections 4 and 5 contain a description of our algorithm.

2. BASIC DEFINITIONS AND FACTS

Let Σ be an alphabet. Following standard notation, we use Σ^+ to denote the free semigroup generated by Σ, and set $\Sigma^* = \Sigma^+ \cup \{\lambda\}$, where λ is the empty word. An element of Σ^+ is called a *string* or *word*, and is denoted by one of the letters l, s, u, v, w, x, y and z. If $x = vwy$, then the integer $|v|+1$, where $|v|$ is the *length* of v, is the *(starting) position* in x of the *substring* w of x. We also say that each one of the positions $|v|+1, |v|+2, ..., |v|+|w|$ of x is *covered* by the *occurrence* of w *at* position $|v|+1$.

A word x is *primitive* if setting $x = s^k$ implies $k = 1$. A word x is *strongly primitive* or *square-free* if every substring of x is a primitive word. A *square* is any string of the form ss where s is a primitive word. For example, *cabca* and *cababd* are primitive words, but *cabca* is also strongly primitive, while *cababd* is not, due to the square *abab*. Given a square ss, s is the *root* of that square and also its *period*. Let now w be a substring of x having at least two distinct occurrences in x. Then, there are words u, y, u', y' such that $u \neq u'$, and $x = uwy = u'wy'$. Assuming w.l.o.g. $|u| < |u'|$, we say that those two occurrences of w in x are *disjoint* iff $|u'| > |uw|$, *adjacent* iff $|u'| = |uw|$ and *overlapping* if $|u'| < |uw|$. Then, it is not difficult to show (see, e.g., [14]) that word x contains two overlapping occurrences of a word $w \neq \lambda$ iff x contains a word of the form *avava* with $a \in \Sigma$ and v a word. We can thus list the following

Fact 1. A word x contains a square if and only if some pair of identical substrings of x are adjacent or overlap.

Given a word $x = x_1 x_2 ... x_n$, the i-th *rotation* of x ($i = 1, 2, ..., n$) is the string $w = x_i x_{i+1} ... x_n \; x_1 \; x_2 \; ... \; x_{i-1}$. Since all rotations of x have equal length, then for any two such rotations w and w', $w \neq w'$ implies that w and w' differ in at least one symbol. The following easy fact holds (cf., e.g., [12]).

Fact 2. String x has a total of $|x|/q$ distinct rotations if and only if $x = v^q$ for some primitive word $v \in \Sigma^+$.

3. QUASIPERIODS

A primitive string w is a *period* of another string z if $z = w^c w'$ for some integer $c > 0$ and w' a prefix of w. A string z is *periodic* if z has a period w such that $|w| \leq |z|/2$. It is a well known fact of combinatorics on words that a string can be periodic in only one period [13].

A string w *covers* another string z if every position of z is covered by some occurrence of w in z. In particular, every string is covered by itself. If z is covered by $w \neq z$, we say that z is *quasiperiodic*, and the ordered sequence of all occurrences of w in z is called the *w-cover* of z. A periodic string is always also quasiperiodic, but the converse is not true. A string z is *superprimitive* if z is not quasiperiodic. Clearly, a superprimitive string is also primitive. However, the converse is not true. For example, *aba* is superprimitive and also primitive, but *abaabaab* is primitive but not superprimitive, since the superprimitive string *abaab* covers it. Clearly, for any string z there is always some superprimitive string w that covers z. String w is a *quasiperiod* for z. The following lemma entitles us to speak unambiguously of *the* quasiperiod of a string.

Lemma 1. Every string z has exactly one quasiperiod.

Proof. The assertion is obviously true if z is superprimitive, thus we assume henceforth that z is quasiperiodic. Let w and w' be two distinct quasiperiods for z, and assume w.l.o.g. that $|w'| > |w|$. Since both w and w' cover z, then w must be a prefix of w'. For the same reason, w must be a suffix of w'. We now show that w covers w', thus contradicting the assumption that w' is superprimitive. In fact, let z' be the longest common prefix of z and w' such that w covers z'. Assume then $|z'| < |w'|$. Since w covers z, then $|z'| \geq |w'| - |w|$. But w is a suffix of w', whence w also covers w'. •

An occurrence of a string w in another string z is called henceforth a *(w-)segment*, and is identified by the pair (i, w), where i is its starting position in z, or simply by i when this causes no confusion.

Lemma 2. Let w be the quasiperiod of z, and i and j be two consecutive segments in the w-cover of z. Then $v = z_i z_{i+1} ... z_{j-1}$ is a primitive word.

Proof. Assume that v is not primitive and set $v = u^c$ for some primitive word u and $c > 1$. Under our assumptions, we have an occurrence of $v^2 = u^{2c}$ at position i in z. But then there is an occurrence of v at position $i + |u| < j$, whence i and j are not consecutive segments in the w-cover of z, a contradiction. •

Corollary 1. Under the hypothesis of Lemma 2, string $v = z_i z_{i+1} \ldots z_{j-1}$ is the root of a square in z.

Proof. Straightforward. •

Let now x be a string of n symbols. A segment (i, z) of x such that z is quasiperiodic *spans* a *quasiperiodicity* of x. A quasiperiodicity z of x is fully identified by the triplet of its starting position i in x, its quasiperiod w and its *span* $|z|$. Quasiperiodicity $(i, w, |z|)$ of x is *maximal* if the following two conditions are satisfied. First, there is no other quasiperiodicity $(i', w, |z'|)$ of x such that $|z'| > i - i' + |z|$. In other words, $(i, w, |z|)$ is not embedded in another quasiperiodicity having identical quasiperiod. Second, letting a be the symbol of x at position $i + |z|$, we have that wa does not cover za. Clearly, any two maximal quasiperiodicities in the form $(i, w, |z|)$ and $(i', w, |z'|)$ must be disjoint.

We are interested in detecting all maximal quasiperiodicities of a string x. Our approach will be similar to the one adopted in [6] for detecting all squares in x. In particular, we resort to the notion of a *suffix tree* for x [17] (see Fig. 1). Informally, the suffix tree T_x associated with string x is a digital search tree that collects all suffixes of xb, where b is a symbol not belonging to Σ. In the compact representation of the tree, each arc of T_x is labeled with a substring of x and each leaf is labeled with the starting position of a unique suffix of xb. Thus, the concatenation of the labels on the (unique) path leading from the root of T_x to leaf i describes the suffix of xb starting at position i. The label of each arc can be compactly encoded into a suitable pair of pointers to a single reference copy of x. Thus, T_x can be stored in space linear in $|x|$. The construction of T_x for a string x of n symbols can be carried out in time $O(n \log |\Sigma|)$ [17].

Following [17], we say that a substring w of x has a *proper locus* in T_x if there is a node α of T_x such that the concatenation of the labels from the root of T_x to α describes w. It is easy to check that, if a substring w of x has no proper locus in T_x, then there is always at least another substring w' in the form $w' = wv$ which does. The proper locus of the shortest such *extension* w' of w is the *extended locus* of w. In the following, we say that α is the *locus* of w in x to indicate that α is either the locus or the extended locus of w, and we use T_x^α to denote the subtree of T_x rooted at α.

Lemma 3. Let w be a substring of x and let α be the locus of w in T_x. Then, the leaves of T_x^α are the starting positions of all and only the occurrences of w in x.

Proof. An immediate consequence of the definition of T_x. •

Lemma 4. Let $(i, w, |z|)$ be a maximal periodicity of x. Then, w has a proper locus in T_x.

Proof. Assume that w has only an extended locus in T_x, and let w' be the extension of w such that the locus α of w' is the extended locus of w. Let $i_1, i_2, ...i_k$ be the ordered sequence of w-segments that cover z. Observe that $z = x_{i_1}x_{i_1+1}...x_{i_k+|w|-1}$. By Lemma 3, we have that every occurrence of w in x is a prefix of a corresponding occurrence of w'. Consider the substring $z' = x_{i_1}x_{i_1+1}...x_{i_k+|w'|-1}$. Clearly, w' covers z'. But z is a prefix of z', whence periodicity z is not maximal. •

Let $\mathcal{S}$ be an arbitrary set of w-segments of x, ordered according to their starting positions. A maximal substring $\mathcal{N}$ of $\mathcal{S}$ such that any two consecutive segments of $\mathcal{N}$ are either adjacent or overlap is called a *run* [7]. The *size* of a run is the number of segments in it. When reasoning in terms of a node α of T_x, we use $W(\alpha)$ to denote the word w having node α as its proper locus, and we use $d(\alpha)$ to denote the *depth* of α, defined as $d(\alpha) = |w| = |W(\alpha)|$. A maximal substring $i_1, i_2, ..., i_k$ $(k > 1)$ of the ordered sequence of leaves in T_x^α with the property that, for $1 < f \leq k$, $i_f - i_{f-1} \leq |W(\alpha)| = d(\alpha)$ represents a run of $W(\alpha)$ segments based on the set of all $W(\alpha)$ segments. We say that $\mathcal{N}$ is a run *at* α. A run $\mathcal{N}$ *coalesces* at α if $\mathcal{N}$ is a run at α but $\mathcal{N}$ is not a run at any of the children of α.

Theorem 1. $(i, w, |z|)$ is a maximal quasiperiodicity of x if and only if there is a node α in T_x and a run $\mathcal{N} \equiv \{i_1, i_2, ..., i_k\}$ coalescing at α such that $i_1 = i$, $i_k = i - 1 + |z| - d(\alpha)$ and for no ancestor β of α leaf i_1 falls in the same run at β with leaf $i_1 + d(\alpha) - d(\beta)$.

Proof. (*if*). Let α be a node of T_x and $\mathcal{N} \equiv \{i_1, i_2, ...i_k\}$ be a run that coalesces at α and having the properties stated in the claim. Clearly, the segment (i_1, z) that corresponds to $x_{i_1}x_{i_1+1}...x_{i_k-1+d(\alpha)}$ spans a quasiperiodicity of x. Let $x_{i_k+d(\alpha)} = a$. Since $\mathcal{N}$ coalesces at α, then word $W(\alpha)a$ cannot cover za. Thus the only way in which $(i, W(\alpha), |z|)$ could fail to be maximal is if $W(\alpha)$ is not superprimitive. Assume that this is the case and let y be the quasiperiod of $W(\alpha)$. It is easy to see that, since $W(\alpha)$ has a proper locus in T_x, then so does the suffix y of $W(\alpha)$. Since y is also a prefix of $W(\alpha)$, then the proper locus β of y is an ancestor of α. Clearly, i_1 and $i_1 + |z| - 1 - |y|$ share a run at β, contrary to the assumption.

(*only if*). By Lemma 4, w has a proper locus α in T_x. Since $(i, w, |z|)$ is maximal, then the occurrences of w that cover z form a run $\mathcal{N}$ at α. Assume that $\mathcal{N}$ does not coalesce at α. Then, there is a direct son γ of α such that $\mathcal{N}$ is also a run at γ. But then, every segment (j, w) in the cover of (i, z) can be extended into a corresponding segment

(j, w') where $w' = wv = W(\gamma)$. Letting a be the first symbol of v, we have then that wa covers za, which contradicts the hypothesis that $(i, w, |z|)$ be maximal. Assume now that for some ancestor β of α leaves i and $i + d(\alpha) - d(\beta)$ fall in the same run. Then $W(\beta)$ covers w, which contradicts the assumption that w is superprimitive. •

4. CLIMBING T_x WITH RUNS

Based on Theorem 1, the task of detecting all maximal quasiperiodicities in x can be divided into two subtasks. The first subtask consists of computing all runs that coalesce at the internal nodes of T_x. The second subtask is to check, for each one of such runs, whether or not its constituent segments are superprimitive. Although we shall see that these two subtasks can be both carried out during a single walk through T_x, it is convenient to consider them separately.

In this section, we concentrate on the implementation of the first subtask, i.e., the computation of all saturating runs of T_x. Such a computation will be carried out during a bottom-up visit of T_x, in such a way that the synthesis of the runs that coalesce at the generic node α is based somewhat on the already computed runs at the children of α. The crux of our method is to maintain an appropriate description of the collection of runs at each node of T_x as we climb up from the leaves towards the root of the tree. The computation of all saturating runs in T_x is a trivial by-product of this maintenance. In the following, we assume for simplicity of exposition that T_x is a binary tree, but it will be apparent that this restriction can be waived with no substantial penalty. Throughout the rest of this section, we concern ourselves with establishing the following result.

Theorem 2. There is an algorithm to detect all saturating runs of x in $O(n \log^2 n)$ time.

Proof. The proof is a consequence of the explicit construction that follows. •

Our maintenance scheme consists of repeated applications of two basic procedures. The first such procedure is called *MERGE* and operates as follows. Let α be a node in T_x, and let α_1 and α_2 be the children of α. Let L_{α_1} and L_{α_2} be the sorted lists of leaves at α_1 and α_2, respectively, and assume that L_{α_1} and L_{α_2} are individually partitioned into disjoint consecutive sublists where each sublist represents a run of $W(\alpha)$ segments. The task of *MERGE* at α is to combine the structured lists L_{α_1} and L_{α_2} into a single, similarly structured list L_α. Thus, each sublist of L_α will be a run of $W(\alpha)$ segments at α, and the runs saturating at α will be given precisely by the sublists of L_α that did not formerly exists in either L_{α_1} or L_{α_2}. Note that the input to this *MERGE* consists of segments of length $d(\alpha)$, while the runs at, say, α_1 contain segments of length $d(\alpha_1) > d(\alpha)$. In other words, the input to the *MERGE* at α does not exactly coincide with the outputs of the two

*MERGE*s that took place at α_1 and α_2, respectively. In fact, the list that results, say, from the *MERGE* at α_1 is partitioned into runs of $W(\alpha_1)$ segments, while the corresponding list L_{α_1} that serves as input to the *MERGE* at α must be partitioned in terms of $W(\alpha)$ segments. The transformation of one partition into the other is the objective of the second procedure, which is called *CLIP*. In general, *CLIP* transforms the run partition of the list L_α of all $W(\alpha)$ segments at node α into the run partition that would pertain to L_α if this list consisted of $W(\beta)$ segments, where $\beta = Father[\alpha]$. Clearly, an appropriate mixture of *MERGE*s and *CLIP*s will realize the evolution of the runs as we climb up in T_x from the leaves to the root. The time performance of our maintenance scheme depends on the implementation of *MERGE* and *CLIP*. This is examined next.

We consider first the *MERGE* at the generic node α. Let as before L_{α_1} and L_{α_2} be the sorted lists of leaves at α_1 and α_2, respectively, and assume w.l.o.g. that $|L_{\alpha_1}| \leq |L_{\alpha_2}|$. We can obtain the sorted list of leaves $L_\alpha = L_{\alpha_1} \cup L_{\alpha_2}$ in time $O(|L_{\alpha_1}| \log |L_{\alpha_2}|)$ by standard balanced-tree [1] allocation of each list. Actually, the data structure introduced in [6] can support the computation of the sorted lists of leaves at all nodes of T_x in overall $O(n \log n)$ time. Recall, however, that the *MERGE* at α must also produce the partition of L_α into runs of $W(\alpha)$ segments, starting from the appropriate partitions of L_{α_1} and L_{α_2}. Therefore, we actually need to allocate each run of a list into a separate balanced tree. The global balanced tree considered at the beginning is needed mainly as an index to access the individual runs and unaggregated segments in the list. In conclusion, the data structure needed at node α can be visualized as a collection of balanced trees organized on two levels. At the top level, we have the *index* tree, each leaf of which is either a simple $W(\alpha)$ segment or the representative (e.g., the leftmost $W(\alpha)$ segment) of a run, according to the case. At the bottom level, each leaf of the index that represents a run points to a balanced tree specifically dedicated to allocate that run. In the following, we use the term *two-tree* to refer to this structure. As seen earlier, the global lists only undergo expansions as the bottom up computation progresses. On the other hand, we will see that the individual run sublists behave like *concatenable queues*, i.e., they undergo *concatenations* as well as *splittings* (we conform to [1] for these notions). This and other circumstances that will be seen in the sequel prevent the resort to the technique of [6]. However, we are already in a position to show that the total work charged by the *MERGE*s can be bounded by $O(n \log^2 n)$. For this, it is sufficient to perform a *MERGE* by always inserting segments from the smaller list into the larger one. Then, each leaf can be involved in at most $\log n$ insertions. If two-trees are used to implement the lists of segments with their corresponding run partitions, then each insertion charges $O(\log n)$ elementary steps. In fact, at any given time there can be at most n segments in any index list or individual run, and the insertion of a newcomer segment only involves a finite number of concatenable-queue operations on balanced trees. In conclusion, the total cost of all applications of our current version of

MERGE can be bounded by $O(n \log^2 n)$.

Consider now the task of *CLIP*. Each segment at the outset of a *CLIP* can be regarded as the clipped version of a corresponding segment in the input. If we imagine to perform such a clipping simultaneously on all segments at α, we can expect in general that the clipping of a segment originally in a run will break the run into two pieces. Thus, it appears that, in order to extract the run partition at $Father[\alpha]$, we would have to re-scan the list L_α of $W(\alpha)$ segments clipping the segments one by one. However, assume that the clipped segment $(i, W(Father[\alpha]))$ breaks a former run of $W(\alpha)$ segments at α. This implies that i is the position of a substring ss of x such that $i + |s|$ is the successor of i in T_x^α and $d(Father[\alpha]) < |s| \leq d(\alpha)$. The following known fact from [6] proves that s is primitive, whence ss is a square in x.

Fact 3. (j, yy) is a square in x if and only if there is a node γ in T_x such that $d(\gamma) \geq |y|$, and j and $j + |y|$ are consecutive leaves in T_x^γ.

In conclusion, a run is not affected in the transition from α to $Father[\alpha]$ unless such a transition splits the run in correspondence with a pair $(i, i + |s|)$ of consecutive $W(\alpha)$ segments such that $|s|$ is the period (i.e., root length) of a square (i, ss) in x and $d(Father[\alpha]) < |s| \leq d(\alpha)$. If one could access in succession only those leaves of L_α that are starting positions of such squares, then the update of the run partition of L_α would only require the time necessary to perform a number of run splittings equal to the number of these leaves. Each splitting can be charged to the unique square it destroys. Thus, the total number of splittings performed throughout the bottom-up computation is bounded by the maximum possible number of squares in x i.e., $O(n \log n)$. If we manage to implement our *CLIP*s using the two-tree allocation of runs discussed earlier, then each split will charge $O(\log n)$ time, whence the total work charged by all *CLIP*s throughout our bottom-up computation will be $O(n \log^2 n)$. Before we can claim this bound, however, we need to analyze the overhead imposed by the maintenance of the lists which provide fast sequential access to the needed positions of squares. We call these lists *access* lists. At node α, we will have one access list for every distinct square period $p < d(\alpha)$. The idea is that, in the transition from α to $Father[\alpha]$, we will individually process, in order of decreasing periods, the access lists relative to the periods larger than $d(Father[\alpha])$. An access list is discarded after this use.

Observe that Fact 3 guarantees that all positioned squares (and periods) needed in the transition from α to $Father[\alpha]$ are detected from pairs of consecutive adjacent or overlapping segments during one of the *MERGE*s that take place at α or at some descendant γ of α. As part of that *MERGE*, we may have that the first term in each newly discovered pair of adjacent or overlapping segments be inserted into its appropriate access list. To

ensure the correctness of our approach, we need then to show that every segment from L_γ that ends up in one of the access lists at α has the same successor in L_α than it had in L_γ. This is done in the following Lemma 5. The lemma also ensures that a leaf of L_α cannot simultaneously belong to two or more access lists, whence the total number of entries in the collection of all access lists at any given node α is linear in the number of leaves in T_x^α.

Lemma 5. Let γ be a descendant of α in T_x, and let j and $j + |v|$, with $|v| < d(\alpha)$ be consecutive leaves in L_γ. Then j and $j + |v|$ are consecutive leaves in L_α.

Proof. By Fact 3 (j, vv) is a square, and thus v is primitive. By Fact 2, all rotations of v are distinct. Assume the existence of an intermediate node β on the path from γ to α such that a leaf h, with $j < h < j + |v|$, is in T_x^β. Then there is an occurrence of v at h. But this is impossible, since the segments of length $|v|$ that begin between j and $j + |v|$ are precisely all distinct rotations of v. •

To summarize, consider again the list L_α of all leaves in T_x^α. Let $p_1 > p_2 > ... > p_k$ be the distinct periods of squares the roots of which have loci at α or at ancestors of α in T_x. Finally, assume that, for each p_f $(1 < f < k)$, we have the ordered list L_α^f of all leaves of L_α that are starting positions of squares having period length p_f. The implementation of *CLIP* in the transition from α to $Father[\alpha]$ is as follows. We consider, in order of decreasing period, all the lists L_α^f such that $p_f > d(Father[\alpha])$. For each such list, we scan the leaves in the list in succession and, in correspondence with each such leaf, split a former run at α. When all lists that needed to be considered are exhausted, they are simply discarded, having produced the run partition at $Father[\alpha]$. As mentioned, the total charges made by this work thru all *CLIP*s are $O(n \log^2 n)$. In fact, each run splitting that takes place during a *CLIP* can be charged to a distinct square among the $O(n \log n)$ squares of x, and each run splitting takes $O(\log n)$ steps. It is not difficult to upgrade *MERGE* in such a way that the procedure also maintains the L_α^f access lists, without penalty in the time complexity of the procedure.

In conclusion, the bottom-up computation of runs described in this section takes $O(n \log^2 n)$ time and requires linear auxiliary space. This concludes our discussion of Theorem 2. In view of Theorem 1, however, our computation only yields all candidate quasiperiodicities in x. In order for one such candidate triplet $(i, w, |z|)$ to actually be a quasiperiodicity, word w must be superprimitive. Thus, our strategy must provide also means for certifying the superprimitivity of all candidate quasiperiods detected. This problem is studied in the next section.

5. THE AUTHENTICATION OF QUASIPERIODS

Recall that, whenever at some node α of T_x a new run $\mathcal{N}$ coalesces, then the segment (i, z) of x spanned by that run is instantaneously known. In fact, i is the first leaf in $\mathcal{N}$ and also the starting position of z, and $|z| = (i_k + |w| - 1) - i$, where i_k is the last leaf in $\mathcal{N}$ and $w = W(\alpha)$ (whence $|w| = d(\alpha)$). With trivial extra bookkeeping, the triplet $(i, |w|, |z|)$, which fully characterizes $\mathcal{N}$, can be produced in constant time during the *MERGE* at α. Note that the format of this triplet is similar to that of a quasiperiodicity, and in fact it denotes a quasiperiodicity if and only if w is superprimitive. In this section, we describe how the superprimitivity of every candidate quasiperiod is tested.

Since there are no more runs than there are squares in x, then we do not have to test more than $O(n \log n)$ candidate quasiperiodicities. Testing the superprimitivity of an isolated string requires at least time linear in the length of that string. However, Theorem 1 suggests that one could exploit the structure of suffix trees to perform each test much faster. As the following brief discussion shows, this is true, but the main problem is to avoid having to test the same candidate too many times.

Assume we made the convention that, at the time that $\mathcal{N}$ coalesces, the corresponding triplet $(i, |w|, |z|)$ is appended to a special *test* list $\mathcal{Q}$ associated with $\mathcal{N}$. Triplets in a test list can be stored in order of increasing i. As said, triplet $(i, |w|, |z|)$ is introduced in the test list $\mathcal{Q}$ associated with $\mathcal{N}$ when $\mathcal{N}$ coalesces, and is removed from a test list if and only if some subsequent *MERGE* proves w to be quasiperiodic. Lemma 5 ensures that, once a triplet is removed from its test list, it is never re-introduced in any test list. In fact, both the introduction of $(i, |w|, |z|)$ in a test list and its possible subsequent removal can be put in one-to-one correspondence with a distinct square in x. Thus, no more that $O(n \log n)$ insertions and deletions of triplets take place throughout the bottom-up visit of T_x. If also the individual test lists are allocated each on a separate balanced tree, then their maintenance thru the bottom-up visit of T_x does not affect the $O(n \log^2 n)$ time bound of the preceding section. In fact these lists are just merged during *MERGE*s and split during *CLIP*s, much in the same way as it happens to their associated runs. However, the real bottleneck along these lines is not in the maintenance of test lists but in their use. After each *MERGE*, we would have to consider explicitly each individual test lists and check its elements one by one for superprimitivity. An element in a list may be checked several times without this resulting in the removal of that element from the list. Thus, the associated work may well exceed the $O(n \log^2 n)$ time bound that we want to achieve. The upgrade of procedure *MERGE* which we now proceed to describe follows a different approach which gets around this difficulty. As a result, we will limit to $O(\log n)$ the number of tests per candidate, and we will be able to perform each test in constant amortized time. From now on, we extend the notion of run to individual segments, which are thus considered

singleton runs.

Our first step is to associate a unique special segment with each node of T_x. (Among other things, we need such a segment to take upon itself some of the work charged by the subsequent superprimitivity tests.) For this, consider a node α of T_x and let, as earlier, L_{α_1} and L_{α_2} be the run-partitioned lists of $W(\alpha)$ segments at the two children α_1 and α_2 of α, respectively. Assume $|L_{\alpha_1}| \leq |L_{\alpha_2}|$, so that the elements of L_{α_1} are inserted in succession into the run structure L_{α_2}. Let $\mathcal{N}$ be the first one among the runs coalescing at α. In the general case, $\mathcal{N}$ results from the coalescence of a number of previously disjoint runs of L_{α_2}, $\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_h$, which were connected by *MERGE* through segments from L_{α_1} (see Fig. 2). Observe that, since $\mathcal{N}$ coalesces at α, then at least one of these runs is nonempty, and thus $\mathcal{N}_1$ is certainly nonempty. Let (i_1, w) and (i_t, w) be the first and last segment of $\mathcal{N}_1$, respectively. Since $\mathcal{N} \neq \mathcal{N}_1$, then if (i_1, w) does not have a predecessor in $\mathcal{N}$ then (i_t, w) must have a successor in $\mathcal{N}$. We define the *characteristic segment* of α to be (i_1, w), if (i_1, w) has a predecessor in $\mathcal{N}$, and to be the successor of (i_t, w) otherwise. Thus, the immediate predecessor of the characteristic segment of every node was not the immediate predecessor of that segment prior to the *MERGE* at that node. Since a characteristic segment and its immediate predecessor are not disjoint at the time of coalescence, then the introduction of characteristic segments has ultimately the effect of mapping each node of T_x into a distinct candidate quasiperiod in x. Obviously, if this quasiperiod fails to be superprimitive, then every candidate quasiperiodicity issued at α also fails, and vice versa.

During the *MERGE* at α, it is trivial to spot the characteristic segment of α on the fly. The most important consequence of the introduction of characteristic segments is the fact, already noted earlier, that we may have only $O(n)$ charateristic segments (one for each node of T_x), even though we could have $\Theta(n \log n)$ candidate quasiperiodicities. This means that we can keep a pointer from every candidate quasiperiodicity to its node of coalescence, and tag nodes that are subsequently discovered not to be loci of superprimitive strings. At the end of the visit of T_x, it is easy to combine the information accumulated in this way, and thus get all and only the maximal quasiperiodicities of x.

Assume now that that (j, w) was found to be the characteristic segment of α and let $(i, w, |z|)$ be the triplet describing the associated run $\mathcal{N}$ and j' be the predecessor of j in $\mathcal{N}$. Observe that, if $j - j' \leq |w|/2$, then w is periodic, hence not superprimitive. We can thus state the following fact.

Fact 4. If $j - j' \leq |w|/2$, then $(i, w, |z|)$ is not a maximal quasiperiodicity.

We know from Theorem 1 that, if w is not superprimitive, then a run covering w will coalesce at some node higher in the tree. The following lemma shows that such a run will be actually a necklace, a condition that is crucial to our construction.

Lemma 6. Assume $j - j' \leq |w|/2$. Then, there is an ancestor β of α such that j' and $j' + |w| - W(\beta)$ are in a same necklace coalescing at β.

Proof. By the definition of characteristic segment, there must be two distinct symbols b and b' in Σ such that (j', wb') and (j, wb) are positioned substrings of x. Since w is periodic, then there is a primitive prefix u of w and an integer $g > 1$ such that $w = u^g u'$, where u' is a possibly empty prefix of u. Clearly, uu' covers w. Moreover, $(j' + |u^{k-1}|, uu'b')$ and $(j + |u^{k-1}|, uu'b)$ are positioned substrings of x. Hence uu' has a proper locus in T_x. Letting β be this proper locus, the claim readily follows. •

Fact 4 and Lemma 6 show that if $j - j' \leq |w|/2$, then we can already predict that a run covering the one just coalesced will be discovered at some later stage. Thus, no action is needed. A run $\mathcal{N}$ for which, for every pair of consecutive segments (l, w) and (m, w) in $\mathcal{N}$, we have $m - l > |w|/2$ is called a *necklace*. Necklaces are the only type of runs that need testing. With easy bookkeeping, it is trivial to check whether a run is a necklace in constant time at the time it coalesces. From now on, we say shorthand that a segment is the characteristic segment of a run or necklace to mean that it is the characteristic segment of the node for which that run or necklace is the first one to coalesce.

Lemma 7. For any position j of x, the number of times that j can be the starting position of the characteristic segment of a necklace is $O(\log n)$.

Proof. Let (j, w) be the characteristic segment of some necklace $\mathcal{N}$, and let (j', w) be the predecessor of (j, w) in $\mathcal{N}$ (see Fig. 3). Let w' be the longest prefix of w for which (j', w') is the characteristic segment of some necklace, and let $\mathcal{N}'$ be this second necklace. Finally, let (j'', w') be the predecessor of (j, w') in $\mathcal{N}'$. It will be sufficient to show that $j - j'' < 2(j - j')/3$. To see this, assume $j - j'' \geq 2(j - j')/3$, as shown in Fig. 3. Then, $|w'| \geq j - j'' \geq 2(j - j')/3$, whence $j'' - j' \leq |w'|/2$. Now, w' is a prefix of w, and thus it occurs at j'. But then w' has a period not exceeding $1/2|w'|$, hence w' is periodic, and $\mathcal{N}'$ is not a necklace. •

We go back to our upgrade of *MERGE*. If (j, w) is found to be the characteristic segment for the first necklace at node α, then the quadruple $(i, j', |w|, |z|)$ describing that necklace is generated and assigned to (e.g., appended to a list associated with) leaf j (cf. Fig. 3). (In practice, we need a pointer from leaf j to node α, and a record of j'; we carry along quadruples for ease in mnemonics.) In our strategy, this quadruple will stay with leaf j throughout a number of *MERGE*s and *CLIP*s, possibly with other quadruples similarly assigned to j in the process.

Assume that $(i, j', |w|, |z|)$ was assigned to leaf j at α but w is not superprimitive. With reference to Fig. 3.b and Fig. 4, we examine the implications of Theorem 1 in this case. By that theorem, there is an ancestor β of α such that letting $w' = W(\beta)$, there are leaves $j'' < j$ and $j''' = j' + |w| - |w'|$ in L_β, possibly with $j'' = j'''$, such that j, j', j'' and j''' are all positions of w' segments in some run (actually, necklace in view of Lemma 6) $\mathcal{N}'$ coalescing at β. Note that as soon as we detect such a situation, then we know that the triplet $(i, w, |z|)$ fails to be a quasiperiodicity, as it is supplanted by the triplet $(i', w', |z'|)$ representing $\mathcal{N}'$.

To better convey our method, we examine first the special situation where the following conditions are simultaneously satisfied (cf. Fig. 3.b):

1) Leaf j'' joins leaves j and j' precisely at β;

2) Leaf j''' coincides with j''; and, finally

3) $j'' + |w'| = j' + |w|$, i.e., the ends of segments (j', w) and (j'', w') coincide.

Assume w.l.o.g. that segment (j'', w') is inserted into a run partition containing segments (j', w') and (j, w'). After the insertion, j'' would find j as its immediate successor in a necklace of L_β. Recall that the quadruple $(i, j', |w|, |z|)$ had been assigned to leaf j. The presence of this quadruple alerts us that an old candidate has to be tested. The test itself is easy, since it only involves checking whether or not j' is in the same necklace as j'' at this point. Note that the quadruple itself is used both to trigger and to shape the test, since it supplies the leaf j'. In general, we would have more than one quadruple associated with j, each coming from some "deeper", still unresolved candidate quasiperiod that also had its characteristic segment at j. All such quadruples can be similarly tested.

In the general case, one or more of conditions 1-3 just discussed will not hold. In particular, leaves j'' and j''' join j at different nodes. Specifically, we will have two significant intermediate nodes between α and β. We call these nodes γ and η, and define them as follows (see Fig. 4). Node γ is the deepest ancestor of α at which leaves j'' and j appear in the same list (note: this is the same as saying that j'' and j appear consecutively in a necklace). Node η is the deepest ancestor of α where leaf j''' appears in the same list with j and j'. Note that we need a way to identify j''' at η. We can have that node γ is an ancestor of η, or $\gamma = \eta$ or γ is a descendant of η. Below, we only illustrate how the case where γ is a descendant of η and $\eta \neq \beta$ is handled by our upgraded *MERGE*. The remaining cases are similar and are left for an exercise. Observe that j''' comes in form of a segment of length $d(\eta) > d(\beta)$, while we need to test whether segments j' and j''' belong in the same necklace at η, i.e., when their corresponding segments have length $|w'| = d(\eta)$. Thus, one difficulty is in computing $d(\beta)$ during the merge at η, and another is in keeping track of the need for a specific test at β in the transition from η to β. Before we proceed with the discussion, we note a lemma that gives a characterization of leaf j''' at η. We will

use such a characterization to overcome the first problem.

Lemma 8. At node η, Assume the integer $\hat{j} = j + |w|$ is inserted in L_η. After such an insertion, the immediate left neightbor of $\hat{j}$ is either j''' or the immediate successor of j'''.

Proof. The claim is a direct consequence of the definition of necklace, i.e., of the fact that $W(\eta)$ is not periodic. •

Let now $W(\gamma) = w'v$. As already observed, the insertion of $(j'', w'v)$ into the necklace partition at γ leads j'' to an impact with leaf j, the carrier of the quadruple $(i, j', |w|, |z|)$. Thus, the procedure learns of the candidate quasiperiodicity $(i, w, |z|)$, and possibly of other candidates attached to the list associated with j. The procedure inserts a spurious (i.e., specially marked) leaf labeled $\hat{j} = j' + |w|$ into L_γ. The role of $\hat{j}$ is to act as a sentinel that awaits the possible arrival of leaf j'''. Similar sentinels are issued for any other quadruple assigned to j.

Consider now the *MERGE* at node η. When j''' joins j and j'', it also finds sentinel $\hat{j}$ in constant time, by virtue of Lemma 8. This is all is needed to compute $|w'| = \hat{j} - j'''$, i.e., the depth of the node β at which the actual test on j' and j''' will have to take place. The procedure stores the triplet $(|w'|, j', j''')$ and a pointer to leaf j''' in a priority queue based on the values of the first term in the triplet. Each element of the priority queue is actually a list containing all triplets that have identical first term. When node β is reached, the procedure is alerted by the presence at the top of the priority queue of triplets having first term $d(\beta) = |w'|$. After constructing the run partition at β, the only thing needed to test triplet $(|w'|, j', j''')$ is knowledge of whether or not j' is smaller than the minimum leaf stored in the run at β containing j'''. This is easily done in $\log n$ time, e.g., by maintaining pointers to the father in two-trees.

The case where η is a descendant of γ is dealt with similarly, except instead of looking for a sentinel in the proximity of of j''' we now look for j''' in the proximity of a sentinel (cf. Lemma 8).

We examine now the performance of the procedure. We can charge each test to the characteristic segment being tested. Equivalently, to the node of T_x that uniquely represents that segment. We have seen that all preparatory stages for a test either take constant time or can be absorbed into the $O(\log n)$ work already charged by the insertion of some leaf. Thus, this preparatory work is absorbed in the $O(n \log^2 n)$ previous global bound. As for the tests themselves (i.e., checking j' and j''' for membership in the same necklace at β), we have already argued that each such test charges $O(\log n)$ steps. Since each node is not tested more than $O(\log n)$ times (cf. Lemma 7), our global bound of $O(nlog^2 n)$ follows.

Theorem 2. There is an algorithm to compute all maximal quasiperiodicities of a string x of n symbols in $O(n \log^2 n)$ time.

References

[1] Aho, A. V., J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Ma. (1974).

[2] Apostolico, A., The Myriad Virtues of Subword Trees, in *Combinatorial Algorithms on Words* (A. Apostolico and Z. Galil, eds.), Springer-Verlag ASI F-12, 85-95 (1985).

[3] Apostolico, A., "Optimal Parallel Detection of Squares in Strings - Part I: Testing Square-freedom" Purdue University TR 932 (1989), *Algorithmica*, to appear. Also, "Part II: Detecting All Squares", Purdue University TR 1012 (1990), submitted for publication.

[4] Apostolico, A., "On Context-Constrained Squares and Repetitions in a String", *R. A. I. R. O. Inf. Theory* **18**, 147-159 (1984).

[5] Apostolico, A. and Z. Galil (eds.), *Combinatorial Algorithms on Words*, Springer-Verlag Nato ASI Series F, Vol. 12, 1985.

[6] Apostolico, A. and F.P. Preparata, "Optimal Off-line Detection of Repetitions in a String", *Theoretical Computer Science* **22**, 297-315 (1983)

[7] Apostolico, A. and F.P. Preparata, "Structural Properties of the String Statistics Problem" *J. Comp. Sys. Sciences* 31, 3, 394-411 (1985)

[8] Brown, M.R. and R. E. Tarjan, A Representation of Linear Lists with Movable Fingers, *Proceedings of the 10-th STOC*, San Diego, Ca., 19-29 (1978).

[9] Crochemore, M. "An Optimal Algorithm for Computing the Repetitions in a Word" *Information Processing Letters* **12**, 5, 244-250 (1981).

[10] Crochemore, M. "Recherche Lineaire d' un Carré dans un Mot" *C.R. Acad. Sc. Paris* **296**, Serie I, 781-784 (1983).

[11] Crochemore, M. and W. Rytter, "Usefulness of the Karp-Miller-Rosenberg Strategy in the Design of Parallel Algorithms", typescript, (1989).

[12] Duval. J.P.. "Factorizing Words over an Ordered Alphabet", *Journal of Algorithms* 4, 363-381 (1983).

[13] Lyndon, R.C. and M.P. Shutzenberger, "The Equation $a^M = c^N c^P$ in a Free Group", *Michigan Mathematical Journal* **9**, 289-298 (1962).

[14] Lothaire, M., *Combinatorics on Words*, Addison Wesley, Reading, Mass., 1982.

[15] Main, M.G. and R.J. Lorentz, "An O(nlogn) Algorithm for Finding all Repetitions in a String", *Journal of Algorithms* **5**, 422-432 (1985).

[16] Main, M.G. and R.J. Lorentz, "Linear-time Recognition of Square-free Strings", in: *Combinatorial Algorithms on Words*, Apostolico, A. and Z. Galil (eds.), Springer-Verlag Nato ASI Series F, Vol. 12, 271-278 (1985).

[17] McCreight, E. M., A Space Economical Suffix Tree Construction Algorithm. *Jour. of the ACM* **25**. 262-272 (1976).

[18] Rabin, M., "Discovering Repetitions in Strings", in: *Combinatorial Algorithms on Words*, Apostolico, A. and Z. Galil (eds.), Springer-Verlag Nato ASI Series F, Vol. 12, 279-288 (1985).

[19] Thue, A., "Über unendliche Zeichenreihen", *Norske Vid. Selsk. Skr. I Mat. Nat. Kl.*, Christiania no. 7, 1-22 (1906).

[20] Thue, A., " Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen", *Norske Vid. Selsk. Skr. I Mat. Nat. Kl.*, Christiania no. 1, 1-67 (1912).

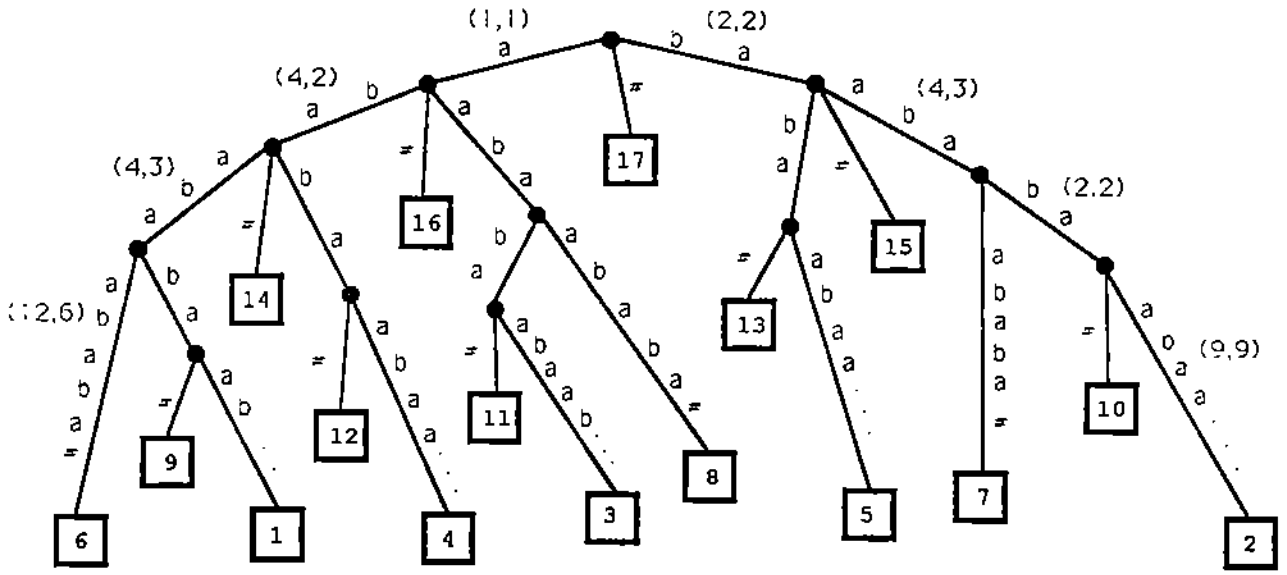


Figure 1

Salient features of a suffix tree.

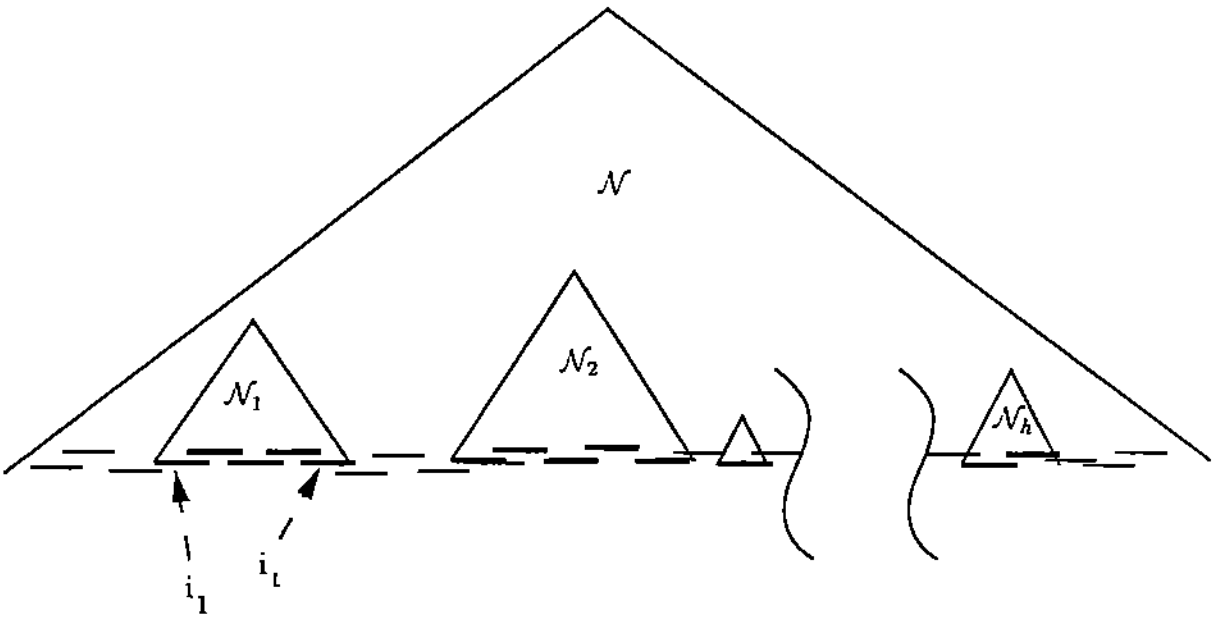
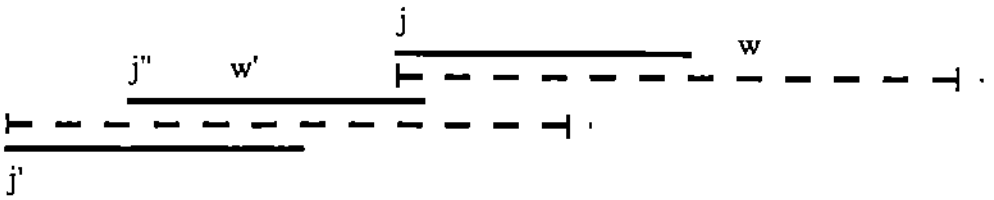
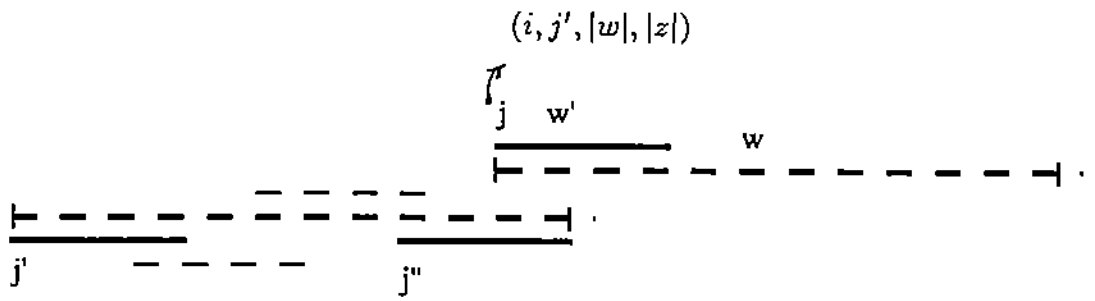


Figure 2

The coalescence of a run $\mathcal{N}$ from runs $\mathcal{N}_1$, $\mathcal{N}_2$, ..., $\mathcal{N}_h$, and segments inserted from L_{α_2}.



a



b

Figure 3

Illustrating Lemma 7.

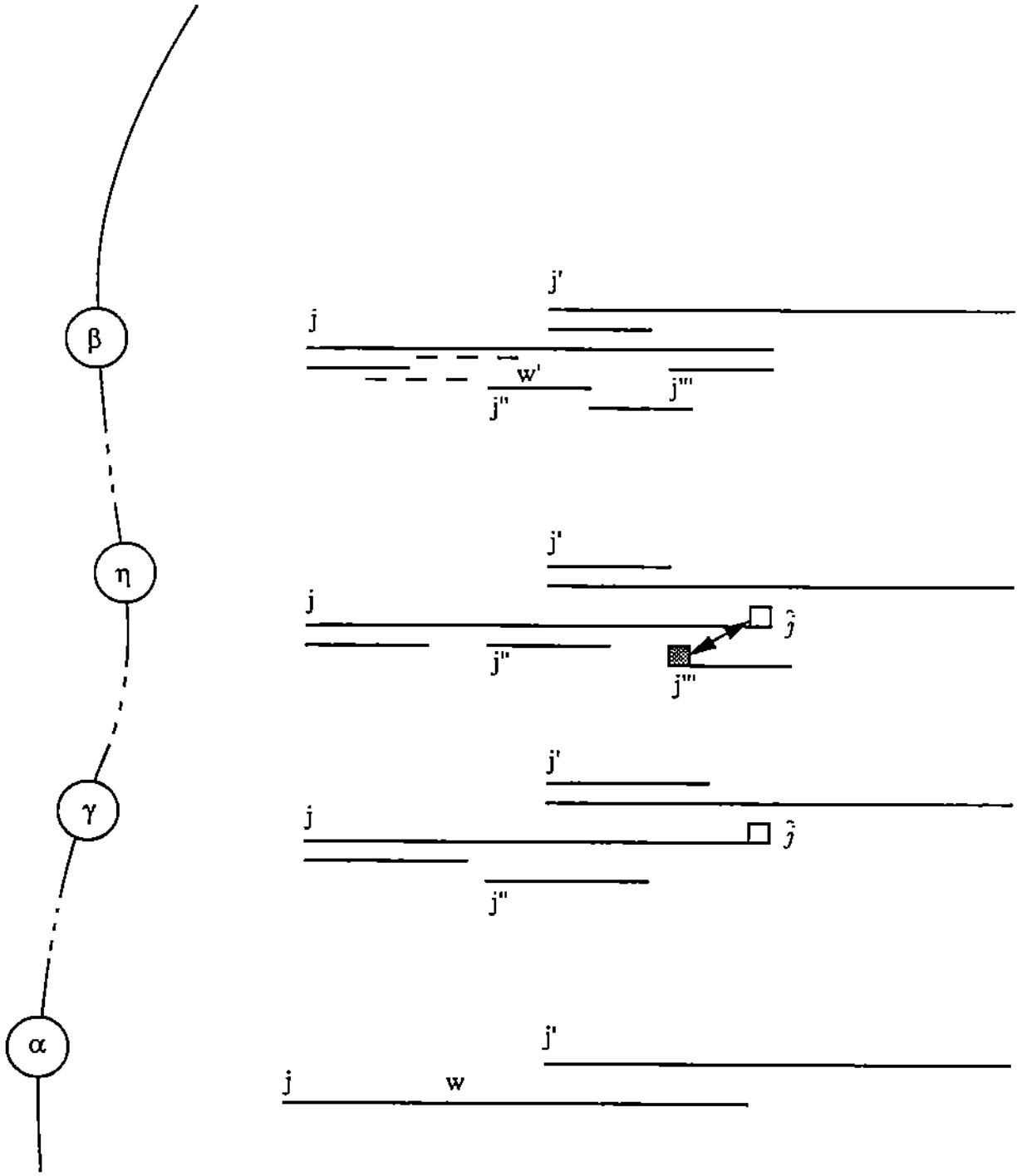


Figure 4

Testing a candidate quasiperiod.